

المملكة المغربية  
Royaume du Maroc  
الوكالة الوطنية لتقنين المواصلات  
Agence Nationale de Réglementation des Télécommunications (ANRT)



المعهد الوطني للبريد و المواصلات  
Institut National des Postes et Télécommunications (INPT)



## Projet de Fin d'Année

Option : Ingénieur des Sciences de Données (Data Engineering)

# Développement d'une application de détection de fraudes en utilisant la bibliothèque de Machine Learning « MLlib » d'Apache Spark

Réalisé par :  
KAMAL ADDI  
MOUAD RIALI

Encadré par :  
Mme. DOUNIA ZAIDOUNI

Département Mathématiques, Informatique et Réseaux  
Année Universitaire : 2020 - 2021

# Remerciement

Tout d'abord, nous tenons à remercier notre professeur Madame Dounia ZAIDOUNI, Enseignante chercheuse à l'institut national des postes et télécommunications, pour son encouragement, ses conseils et sa disponibilité tout au long de ce projet et pour nous avoir accordé toute la confiance nécessaire pour élaborer ce projet librement.

# Abstract

Financial fraud detection is a challenge that must be taken seriously. Although fraudulent transactions are rare and represent only a very small fraction of the activity within an organization. However, a small percentage of the activity can quickly turn into significant financial losses and negative reputational effects on the organization. It is for this reason that effective fraud detection mechanisms play an essential role in protecting the interests of organizations against these negative effects. The primary objective of this work is to develop an application for detecting fraudulent transactions using the Apache Spark MLlib library, then compare the different fraud detection algorithms. We used a synthetic dataset generated using a simulator (Pysim) based on a sample of actual transactions pulled from a month's financial logs from a mobile money service implemented in an African country. We worked on five algorithms : Logistic Regression, Decision Tree, Random Forest, Gradient-Boosted Tree and Naive-Bayes. Performance analysis proved that the Gradient Boosted Tree algorithm is the best among these algorithms.

# Résumé

La détection des fraudes bancaire est un défi qui doit être prise au sérieux. Pourtant, les transactions frauduleuses sont rares et ne représentent qu'une très petite fraction de l'activité au sein d'une organisation. Néanmoins, un faible pourcentage de l'activité peut rapidement se transformer en des pertes financières importantes et des effets néfastes sur la réputation de l'organisme. C'est pour cette raison que des mécanismes efficaces de détection de fraudes jouent un rôle essentiel en protégeant les intérêts des organisations contre ces effets négatifs. L'objectif premier de ce travail est de développer une application de détection de fraude dans les transactions bancaires en utilisant la librairie MLlib d'Apache Spark, puis faire une comparaison entre les différents algorithmes de détection de fraude. Nous avons utilisé un jeu de données synthétiques généré à l'aide d'un simulateur (Pysim) sur la base d'un échantillon de transactions réelles extraites d'un mois de journaux financiers d'un service d'argent mobile mis en œuvre dans un pays Africain. Nous avons travaillé sur cinq algorithmes : Régression logistique, Arbre de décision, Forêt aléatoire, Gradient-Boosted Tree et Naive-Bayes. Les analyses des performances ont montré que l'algorithme Gradient Boosted Tree est le meilleure parmi ces algorithmes.

# Table des matières

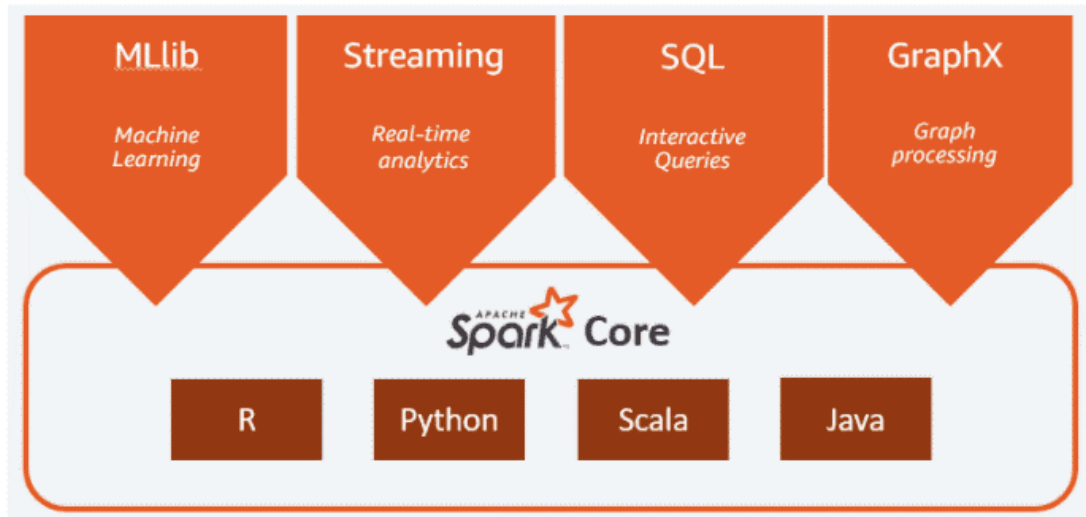
<b>Remerciement</b> . . . . .	<b>1</b>
<b>Abstract</b> . . . . .	<b>2</b>
<b>Résumé</b> . . . . .	<b>3</b>
<b>Introduction générale</b> . . . . .	<b>5</b>
<b>1 Description de l'ensemble de données :</b> . . . . .	<b>7</b>
1.1 Les données financières : . . . . .	7
1.2 Description des variables : . . . . .	7
<b>2 Prétraitement de données :</b> . . . . .	<b>9</b>
2.1 Vérification des données manquantes : . . . . .	10
2.2 Transformation de données : . . . . .	10
<b>3 L'analyse exploratoire de données :</b> . . . . .	<b>13</b>
3.1 Statistique descriptive et visualisations : . . . . .	13
3.2 Analyse de corrélation : . . . . .	16
<b>4 Créer une application de Machine Learning Apache Spark MLlib :</b> . . . . .	<b>18</b>
4.1 Choix de modèle d'apprentissage : . . . . .	18
4.2 Préparer les données : . . . . .	19
4.3 Préparer le modèle d'apprentissage automatique : . . . . .	20
4.4 Prédictions et évaluation du modèle : . . . . .	21
<b>5 Résultats et Mesures de Performance :</b> . . . . .	<b>23</b>
<b>Conclusion :</b> . . . . .	<b>27</b>
<b>Références bibliographiques :</b> . . . . .	<b>28</b>

# Introduction générale

Les journaux parlent fréquemment de fraudes commises par des personnes et des organisations. Ces fraudes ont des répercussions énormes et variées. Avec l'explosion des données massives, les banques font désormais appel au Big Data et au Machine Learning pour détecter et prédire ces fraudes. L'objectif de ce travail de recherche est de répondre à la problématique « Comment utiliser la librairie MLlib de machine learning d'Apache Spark pour détecter les fraudes bancaires ».

Afin de simplifier notre démarche, nous avons travaillé sur un problème d'apprentissage supervisé (supervised learning) et avec des données dont la variable à prédire « isFraud » est labellisée en fraude (1) ou non-fraude (0). Le travail s'effectue en Apache Spark à l'aide de la librairie MLlib de l'API Pyspark en utilisant le langage de programmation Python.

Apache Spark est un moteur de traitement de données rapide dédié au Big Data. Il permet d'effectuer un traitement de larges volumes de données de manière distribuée. Apache Spark a été développé comme une alternative plus rapide à Hadoop, Hadoop-MapReduce lit et écrit les données à partir du disque, ce qui ralentit la vitesse de traitement. Apache Spark stocke ces données en mémoire ce qui permet de réduire le cycle de lecture/écriture. Spark a été développé en Scala et propose des APIs pour Java, Python et R. Son architecture distribuée permet d'exécuter les algorithmes d'apprentissage automatique plus rapidement sans compromettre leurs performances.



*Apache Spark framework (source: AWS)*

Apache Spark MLlib est une bibliothèque open source de machine learning d'Apache Spark, Il offre un ensemble d'algorithmes d'apprentissage automatique pour la classification, la régression, le clustering...etc.

Dans cette étude, nous implémentons cinq algorithmes de classification d'apprentissage automatique supervisé à savoir la régression logistique, Naïve-Bayes, Arbre de décision, Forêt aléatoire, Gradient Boosted Tree et mesurer leurs performances.

# Chapitre 1

## Description de l'ensemble de données :

### 1.1 Les données financières :

Les ensembles de données financières sont importants pour de nombreux chercheurs et en particulier pour nous, dans ce travail de recherche sur la détection de fraudes financières. Une partie du problème est la nature intrinsèquement privée des transactions financières, qui conduit à l'absence d'ensembles de données accessibles au public.

Dans ce projet, nous avons utilisé un jeu de données synthétiques généré à l'aide d'un simulateur (Pysim) sur la base d'un échantillon de transactions réelles extraites d'un mois de journaux financiers d'un service d'argent mobile mis en œuvre dans un pays Africain. Les journaux d'origine ont été fournis par une société multinationale, qui est le fournisseur du service financier mobile qui fonctionne actuellement dans plus de 14 pays à travers le monde.

Cet ensemble de données synthétiques est réduit à 1/4 de l'ensemble de données d'origine et il est disponible sur Kaggle.

### 1.2 Description des variables :

Le tableau ci-dessous présente description des variables de l'ensemble de données :



Variable	Type	Description
type	String	type de transaction : CASH-IN, CASH-OUT, DEBIT, PAIEMENT et TRANSFERT.
amount	Double	montant de la transaction en monnaie locale.
nameOrig	String	identifiant de l'émetteur (le client qui a commencé la transaction).
oldbalanceOrg	Double	solde initial de l'émetteur avant la transaction.
newbalanceOrig	Double	nouveau solde de l'émetteur après la transaction.
nameDest	String	identifiant du destinataire (client destinataire de la transaction).
oldbalanceDest	Double	solde initial du destinataire avant la transaction. Notez qu'il n'y a pas d'informations pour les clients commençant par M (commerçants).
newbalanceDest	Double	nouveau solde du destinataire après la transaction. Notez qu'il n'y a pas d'informations pour les clients commençant par M (commerçants).
isFraud	Integer	Il s'agit des transactions effectuées par les agents frauduleux à l'intérieur de la simulation. Dans cet ensemble de données spécifique, le comportement frauduleux des agents vise à tirer profit en prenant le contrôle des comptes des clients et en essayant de vider les fonds en les transférant sur un autre compte, puis en les encaissant du système. 1 si la transaction est frauduleuse, sinon 0.
isFlaggedFraud	String	1 si la transaction a été signalée frauduleuse, sinon 0.

# Chapitre 2

## Prétraitement de données :

Le nettoyage des données est le processus de correction des incohérences dans les données dans le but de générer des données plus organisées et structurées. Très souvent, les données acquises à partir de la source primaire (appelées données brutes) consistent en des informations manquantes, des erreurs typographiques et un format incohérent, etc. Il est presque impossible d'effectuer une quelconque analyse statistique sur de telles données incohérentes non organisées. Par conséquent, avant d'analyser notre jeu de données et d'entraîner les modèles, il convient de faire un prétraitement de ces données pour qu'elles soient compréhensibles par le framework Pyspark. Il est important de noter que Pyspark ne peut travailler que sur des variables numériques. Il est nécessaire donc de transformer les variables catégorielles en numériques.

Après avoir créé un objet SparkSession pour se connecter à spark et télécharger le jeu de données, il est le temps de commencer à comprendre notre jeu de données.

Le jeu de données contient 6362620 exemples, et chaque exemple de cet ensemble de données contient 11 colonnes. où l'avant dernière est l'étiquette de classe (1 pour une transaction frauduleuse, 0 sinon).

step	type	amount	nameOrig	oldbalanceOrig	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
1	TRANSFER	181.0	C1305486145	181.0	0.0	C553264065	0.0	0.0	1	0
1	CASH_OUT	181.0	C840083671	181.0	0.0	C38997010	21182.0	0.0	1	0
1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

only showing top 5 rows

Pour des raisons de capacités de calcul de nos machines nous allons travailler sur la moitié de ce jeu de données (3M d'exemples).

## 2.1 Vérification des données manquantes :

Heureusement, Il n'y a pas de valeurs manquantes dans ce jeu de données :

```

▶ ▶ M↓
### Get count of nan or missing values
from pyspark.sql.functions import isnan, when, count, col

df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).show()

```

step	type	amount	nameOrig	oldbalanceOrig	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	0	0	0	0	0	0	0	0	0	0

## 2.2 Transformation de données :

Il est important aussi de vérifier les types de données avant de les traiter. Nous assurons ainsi l'exactitude et la cohérence de données. Peu importe le type de données utilisées, toutes les enquêtes font l'objet de certaines vérifications. Cette vérification peut se faire à l'aide de la méthode printSchema :

```

▶ ▶ M↓
# Schema of the data
df.printSchema()

```

```

root
|-- step: string (nullable = true)
|-- type: string (nullable = true)
|-- amount: string (nullable = true)
|-- nameOrig: string (nullable = true)
|-- oldbalanceOrig: string (nullable = true)
|-- newbalanceOrig: string (nullable = true)
|-- nameDest: string (nullable = true)
|-- oldbalanceDest: string (nullable = true)
|-- newbalanceDest: string (nullable = true)
|-- isFraud: string (nullable = true)
|-- isFlaggedFraud: string (nullable = true)

```

D'après ce schéma on voit bien que certaines caractéristiques sont interprétées comme des String (par exemple amount, oldbalanceOrig...). On doit transformer ces colonnes en « intiger » et « Double », sinon le modèle

ça ne va pas marche. Dans ce cas on utilise la méthode `cast()` avec `withColumn()` :

```

> M1

from pyspark.sql.types import DoubleType

df = df.withColumn("step", df.step.cast(DoubleType()))
df = df.withColumn("amount", df.amount.cast(DoubleType()))
df = df.withColumn("oldbalanceOrg", df.oldbalanceOrg.cast(DoubleType()))
df = df.withColumn("newbalanceOrig", df.newbalanceOrig.cast(DoubleType()))
df = df.withColumn("oldbalanceDest", df.oldbalanceDest.cast(DoubleType()))
df = df.withColumn("newbalanceDest", df.newbalanceDest.cast(DoubleType()))
df = df.withColumn("isFraud", df.isFraud.cast('int'))

```

Pour les colonnes, `type`, `nameOrig` et `nameDest`, qui contiennent des données catégorielles. Nous devons transformer ces colonnes en valeurs numériques indexées.

```

+-----+-----+-----+
|   type| nameOrig| nameDest|
+-----+-----+-----+
| PAYMENT| C1912850431| M633326333|
| PAYMENT| C761750706| M1731217984|
| PAYMENT| C2033524545| M473053293|
| CASH_OUT| C512549200| C248609774|
| PAYMENT| C460570271| M1653361344|
+-----+-----+-----+
only showing top 5 rows

```

Deux techniques pour les transformer :

- Pour les identifiants 'nameOrig' et 'nameDest', nous avons séparé le premier caractère de la chaîne ('C' ou 'M') du reste pour le transformer en valeurs numérique binaire (0 ou 1).

```

> M1

import pyspark.sql.functions as F

data = (
    data.withColumn('str_orig', F.substring('nameOrig', 1,1))
    .withColumn('num_orig', F.col('nameOrig').substr(F.lit(2), F.length('nameOrig') - F.lit(1)))
)
data = data.drop('nameOrig')
# show df
data.show(5)

```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|step|  type| amount|oldbalanceOrg|newbalanceOrig| nameDest|oldbalanceDest|newbalanceDest|isFraud|str_orig| num_orig|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1.0|CASH_OUT| 181.0|      181.0|         0.0| C38997010|      21182.0|         0.0|     1|      C| 840083671|
| 1.0| PAYMENT| 7861.64|    176087.23|    168225.59| M633326333|         0.0|         0.0|     0|      C| 1912850431|
| 1.0|  DEBIT| 9644.94|     4465.0|         0.0| C997608398|     10845.0|    157982.12|     0|      C| 1900366749|
| 1.0| PAYMENT| 2560.74|     5070.0|     2509.26| M972865270|         0.0|         0.0|     0|      C| 1648232591|
| 1.0| PAYMENT| 1563.82|      450.0|         0.0| M1731217984|         0.0|         0.0|     0|      C| 761750706|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

- Et OneHotEncoder pour les colonnes 'type', 'str\_orig' et 'str\_dest' :

type column :

```

> ML
from pyspark.ml.feature import StringIndexer
type_indexer = StringIndexer(inputCol="type", outputCol="typeIndex")
#Fits a model to the input dataset with optional parameters.
data = type_indexer.fit(data).transform(data)
data = data.drop('type')
data.show(5)

```

step	amount	oldbalanceOrig	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	str_orig	num_orig	str_dest	num_dest	typeIndex
1.0	181.0	181.0	0.0	21182.0	0.0	1	C	8.40083671E8	C	8.40083671E8	0.0
1.0	7861.64	176087.23	168225.59	0.0	0.0	0	C	1.912850431E9	M	1.912850431E9	1.0
1.0	9644.94	4465.0	0.0	10845.0	157982.12	0	C	1.900366749E9	C	1.900366749E9	4.0
1.0	2560.74	5070.0	2509.26	0.0	0.0	0	C	1.648232591E9	M	1.648232591E9	1.0
1.0	1563.82	450.0	0.0	0.0	0.0	0	C	7.61750706E8	M	7.61750706E8	1.0

str\_orig column :

```

> ML
nameOrig_indexer = StringIndexer(inputCol="str_orig", outputCol="nameOrigIndex")
#Fits a model to the input dataset with optional parameters.
data = nameOrig_indexer.fit(data).transform(data).drop('str_orig')
data.show(5)

```

step	amount	oldbalanceOrig	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	num_orig	str_dest	num_dest	typeIndex	nameOrigIndex
1.0	181.0	181.0	0.0	21182.0	0.0	1	8.40083671E8	C	8.40083671E8	0.0	0.0
1.0	7861.64	176087.23	168225.59	0.0	0.0	0	1.912850431E9	M	1.912850431E9	1.0	0.0
1.0	9644.94	4465.0	0.0	10845.0	157982.12	0	1.900366749E9	C	1.900366749E9	4.0	0.0
1.0	2560.74	5070.0	2509.26	0.0	0.0	0	1.648232591E9	M	1.648232591E9	1.0	0.0
1.0	1563.82	450.0	0.0	0.0	0.0	0	7.61750706E8	M	7.61750706E8	1.0	0.0

str\_dest column :

```

> ML
nameDest_indexer = StringIndexer(inputCol="str_dest", outputCol="nameDestIndex")
#Fits a model to the input dataset with optional parameters.
data = nameDest_indexer.fit(data).transform(data).drop('str_dest')
data.show(5)

```

step	amount	oldbalanceOrig	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	num_orig	num_dest	typeIndex	nameOrigIndex	nameDestIndex
1.0	181.0	181.0	0.0	21182.0	0.0	1	8.40083671E8	8.40083671E8	0.0	0.0	0.0
1.0	7861.64	176087.23	168225.59	0.0	0.0	0	1.912850431E9	1.912850431E9	1.0	0.0	1.0
1.0	9644.94	4465.0	0.0	10845.0	157982.12	0	1.900366749E9	1.900366749E9	4.0	0.0	0.0
1.0	2560.74	5070.0	2509.26	0.0	0.0	0	1.648232591E9	1.648232591E9	1.0	0.0	1.0
1.0	1563.82	450.0	0.0	0.0	0.0	0	7.61750706E8	7.61750706E8	1.0	0.0	1.0

On peut dire dès maintenant qu'il s'agit d'un problème de classification binaire où l'objectif est de former un classificateur capable de détecter si une transaction est frauduleuse (classe 1) ou non (classe 0).

# Chapitre 3

## L'analyse exploratoire de données :

L'analyse exploratoire de données permet de traiter un nombre très important de données et de dégager les aspects les plus intéressants de la structure de celles-ci. Dans cette étape, on cherche à mieux comprendre les données en calculant les différentes mesures récapitulatives, telles que la moyenne, la valeur maximale, la valeur minimale et variance...etc, et en faisant des visualisations sur l'ensemble de données.

### 3.1 Statistique descriptive et visualisations :

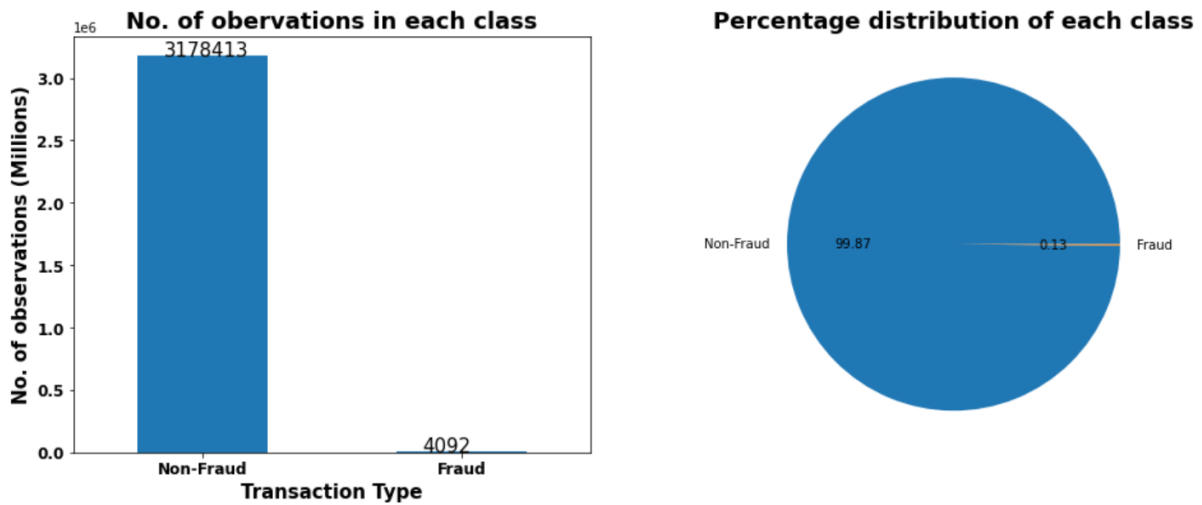
Nous commençons par calculer les mesures récapitulatives pour l'ensemble de données :

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'double' or t[1]=='int']
df.select(numeric_features).describe().toPandas()
```

	summary	step	amount	oldbalanceOrig	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	count	3182505	3182505	3182505	3182505	3182505	3182505	3182505
1	mean	243.32051827098465	180108.63530602978	832749.6707201254	853986.0712083996	1104812.3688028697	1229522.0882875763	0.001285779598146743
2	stddev	142.3514799023828	603080.4075271396	2884824.170356285	2920635.8624181743	3427265.1766493297	3700988.0592913437	0.035834714627960236
3	min	1.0	0.0	0.0	0.0	0.0	0.0	0
4	max	743.0	9.244551664E7	5.958504037E7	4.958504037E7	3.5538143361E8	3.555534163E8	1

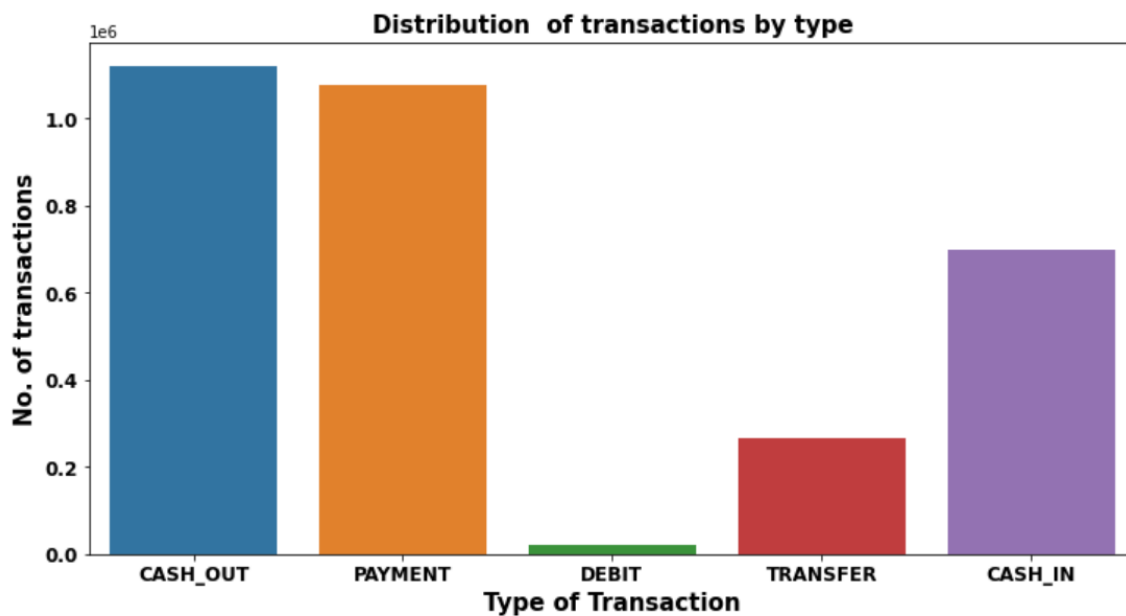
À partir du tableau, nous pouvons observer les intervalles [min, max] des chacune des caractéristiques, les moyennes (step : 243.32 , amount : 180 108.63, oldbalanceOrig : 832 749.67, newbalanceOrig...etc) et les écart-types.

On peut visualiser le nombres de transactions par classe :



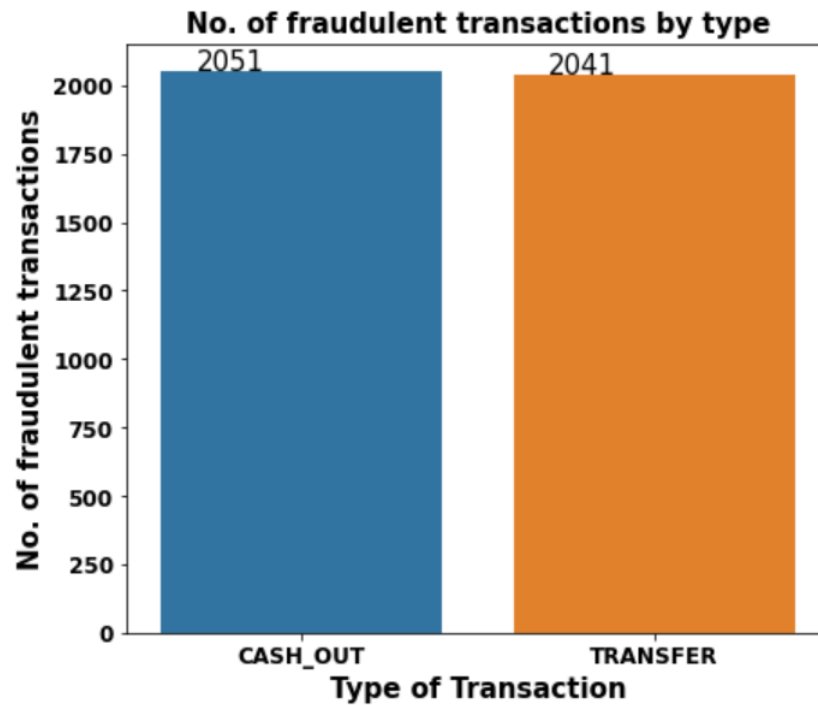
On remarque que les transactions frauduleuses sont rares, (on que 8 213 fraudes (0,13 %)).

On peut visualiser aussi la distribution des transactions par type de transactions :



On remarque ici que la majorité des transactions sont de type CASH\_OUT, PAYMENT et CASH\_IN.

De même la distribution des transactions frauduleuses par type :



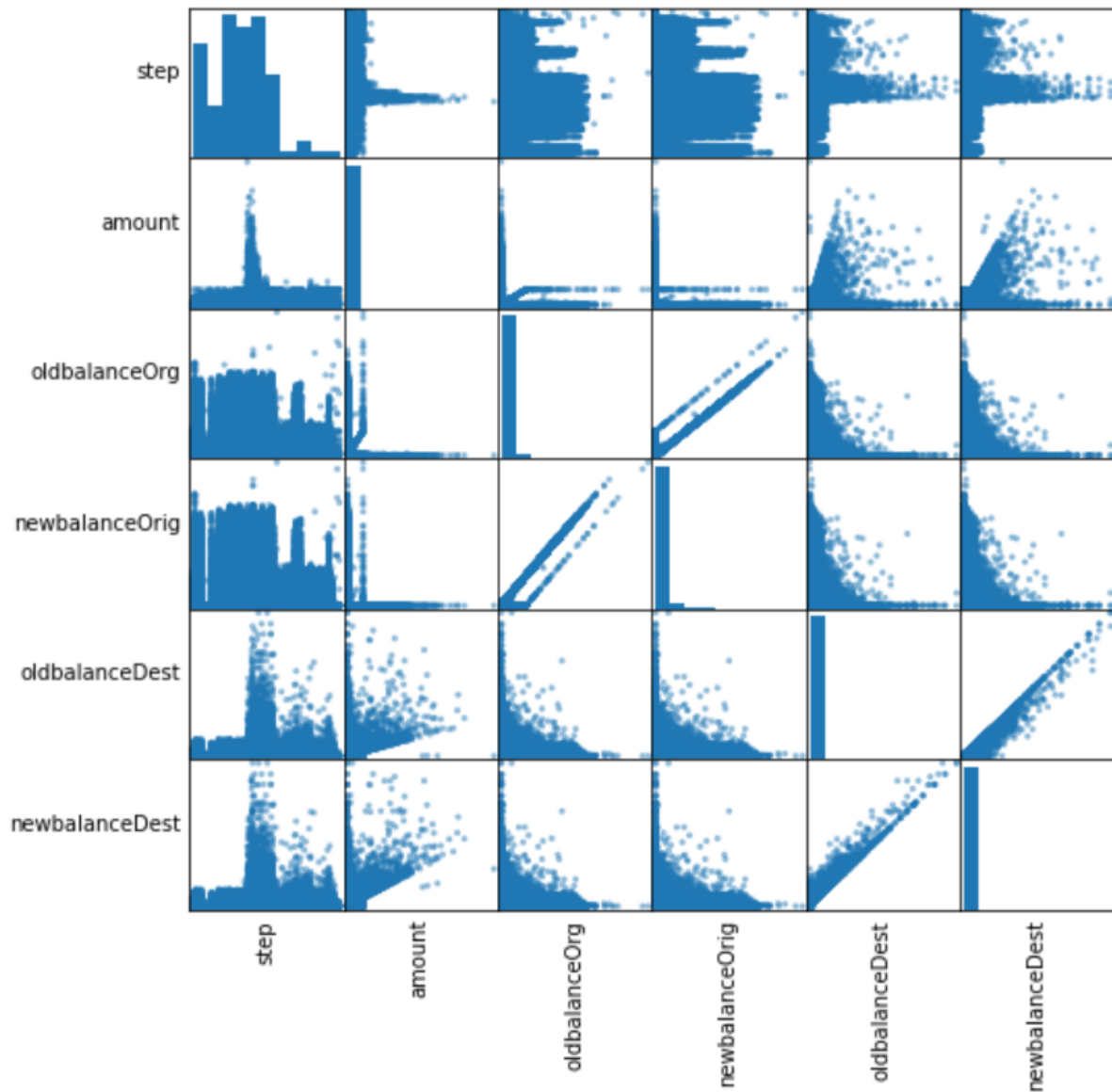
Nous pouvons observer que les transactions frauduleuses sont uniquement des retraits CASH\_OUT et des virements TRANSFER avec des pourcentage de fraude de 50% pour chacun d'entre eux. Pour les autres types, il n'y a pas de fraude.



## 3.2 Analyse de corrélation :

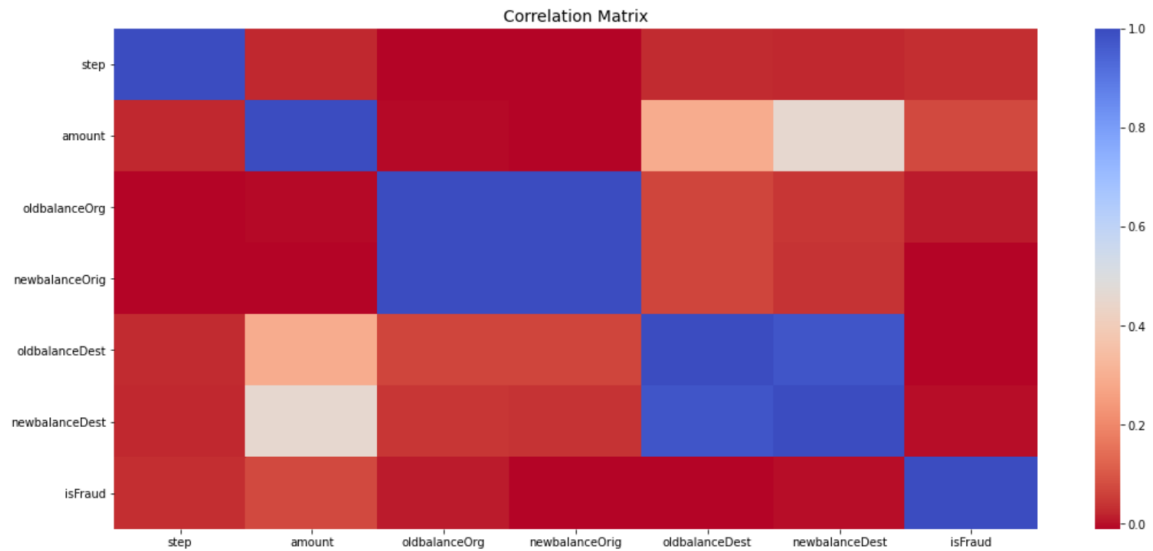
L'analyse de corrélation permet d'étudier les indépendances entre plusieurs variables.

Le figure suivant montre la corrélation entre chacun des variable de notre jeu de données :



On génère aussi la matrice de corrélation entre toutes les variables du dataset :

Le résultat est une table contenant les coefficients de corrélation entre chaque variable et les autres.



D'après la matrice de corrélation, Il n'y a pas de corrélations visibles entre les variables.

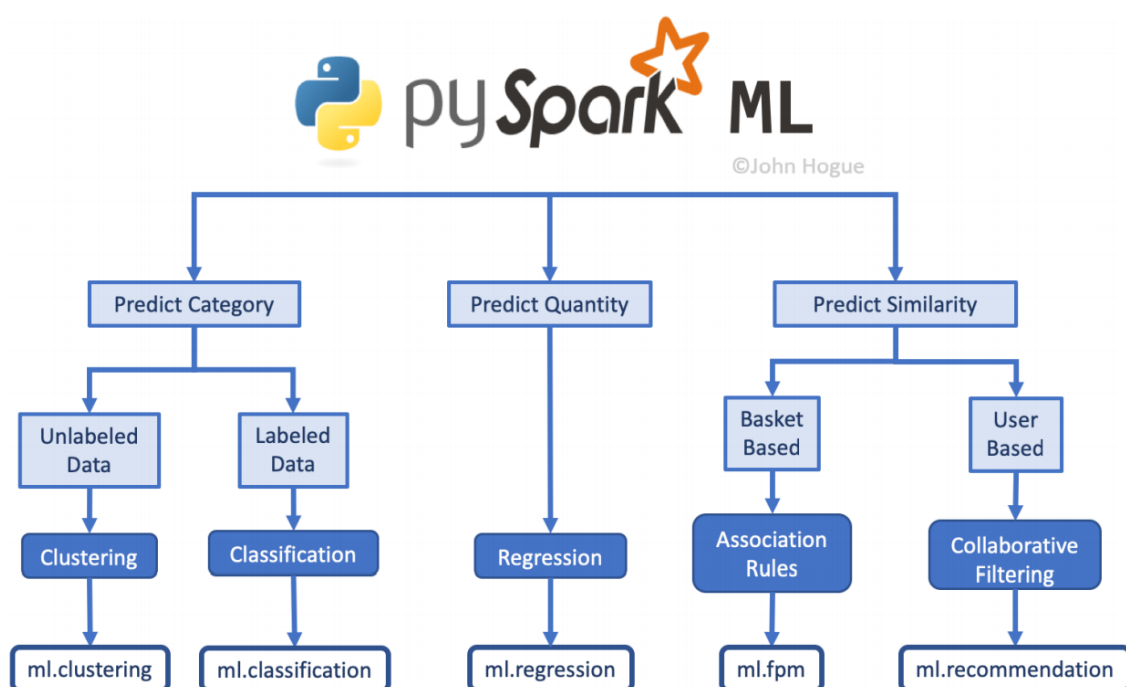
## Chapitre 4

# Créer une application de Machine Learning Apache Spark MLlib :

### 4.1 Choix de modèle d'apprentissage :

Notre objectif est de construire des modèles d'apprentissage automatique pour prédire si une transaction est frauduleuse ou non. La colonne « is%Fraud » est notre variable cible (également appelée label) et le reste des colonnes sont les variables indépendantes (également appelées caractéristiques).

Nous avons des données étiquetées et la sortie de notre modèle est une valeur de la classe 0, 1, donc c'est un problème de classification supervisé.



Dans ce projet on se focalise sur cinq algorithmes parmi les algorithmes de classification supervisés de la librairie MLlib : Régression logistique, Arbre de décision, Forêt aléatoire, Gradient-Boosted Tree et Naive-Bayes.

### 4.2 Préparer les données :

Maintenant que nous avons choisi les algorithmes qu'on va utiliser, nous devons transformer nos données à l'aide de VectorAssembler :

```
ML
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=['amount', 'oldbalanceOrig', 'newbalanceOrig', 'oldbalanceDest',
                                         'newbalanceDest', 'num_orig', 'num_dest', 'typeIndex', 'nameOrigIndex',
                                         'nameDestIndex'], outputCol='features')

data = assembler.transform(data)

ML
data['features', 'isFraud'].show(5)
```

features	isFraud
(10,[0,1,3,5,6],[...]	1
[7861.64,176087.2...	0
[9644.94,4465.0,0...	0
[2560.74,5070.0,2...	0
[1563.82,450.0,0...	0

only showing top 5 rows

Ensuite on divise nos données en ensemble d'entraînement et ensemble de test (80% de training set et 20% de test set).

#### Split the data to training sets and test sets :

```
ML
(train, test) = data.randomSplit([0.8, 0.2], seed=23)

ML
[train.count(), test.count()]

[2544735, 637770]
```

### 4.3 Préparer le modèle d'apprentissage automatique :

Étant donné que nos données sont maintenant dans des ensembles d'entraînement et de test, nous allons créer un modèle d'apprentissage pour l'entraîner sur les données d'entraînement, puis effectuer des prédictions sur les données de test.

Dans cette partie on va expliquer comment implémenter la régression logistique, et ce sera le même workflow pour les autres algorithmes.

On instancier d'abord le modèle en passant en paramètre le label et le vecteur des features, puis on fait l'entraînement avec la méthode `.fit()` sur les données d'entraînement :

#### Logistic Regression :



```
from pyspark.ml.classification import LogisticRegression
#Create a Logistic Regression classifier.
logistic = LogisticRegression(labelCol = "isFraud", featuresCol = "features")
# Learn from the training data.
lrModel = logistic.fit(train)
```

## 4.4 Prédictions et évaluation du modèle :

Évaluer les prédictions de notre modèle est une étape très importante.

Faisons quelques prédictions sur les données de test en utilisant la méthode `transform()` et notre modèle et sauvegardons-les dans `prediction_LR` :

```
▶ ▶≡ ML
```

```
prediction_LR = lrModel.transform(test)
```

```
▶ ▶≡ ML
```

```
prediction_LR.groupBy("isFraud", "prediction").count().show()
```

```
+-----+-----+-----+
|isFraud|prediction| count|
+-----+-----+-----+
|      1|      0.0|    459|
|      0|      0.0| 636923|
|      1|      1.0|    346|
|      0|      1.0|     42|
+-----+-----+-----+
```

Et finalement on évalue le modèle en utilisant les métriques d'évaluation : Recall, Precision, F1 Score, Area Under ROC et Area Under PR.

- Rappel (Recall) - Indique la proportion de positifs réels qui ont été correctement classés. Un modèle qui ne produit aucun faux négatif a un rappel de 1,0.

$$\text{rappel} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}}$$

- Précision - Indique la proportion d'identifications positives (classe 1 prédite par le modèle) qui étaient réellement correctes. Un modèle qui ne produit aucun faux positif a une précision de 1,0.

$$\text{précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

- Score F1 - Une combinaison de précision et de rappel. Un modèle parfait obtient un score F1 de 1,0.

$$F_1 = 2 \times \frac{\text{précision} \times \text{rappel}}{\text{précision} + \text{rappel}}$$

- Area Under ROC (AUC) - L'aire sous la courbe ROC. Un modèle parfait obtient un score de 1,0.

- Area Under PR - L'aire sous la courbe Recall-Precision

```
▶ ▶ ML

tp = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 1)].count()
tn = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 0)].count()
fp = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 1)].count()
fn = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 0)].count()
recall_LR = tp/(tp+fn)
precision_LR = tp/(tp+fp)
f1_score_LR = 2*(recall_LR*precision_LR)/(recall_LR+precision_LR)
print("Recall : ", recall_LR)
print("Precision : ", precision_LR)
print("F1 Score : ", f1_score_LR)
```

```
Recall : 0.4298136645962733
Precision : 0.8917525773195877
F1 Score : 0.5800502933780386
```

```
▶ ▶ ML

# Area under ROC curve
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud")
areaUnderROC_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_LR)
```

```
Area under ROC = 0.9878424845929653
```

```
▶ ▶ ML

# Area under precision-recall curve
areaUnderPR_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_LR)
```

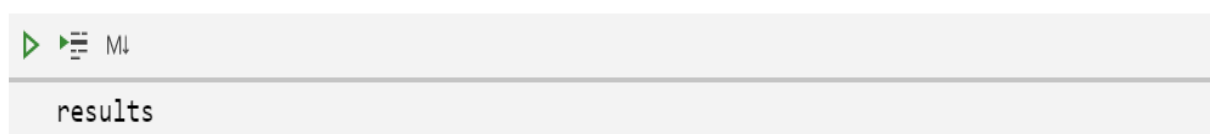
```
Area under PR = 0.47870897117876576
```

# Chapitre 5

## Résultats et Mesures de Performance :

Un bon modèle de Machine Learning, c'est un modèle qui a la capacité à faire des prédictions non seulement sur les données d'entraînement, mais surtout sur de nouvelles données (données de test). C'est pour cela qu'on a fait nos prédictions sur les données de test.

La table suivante représente les différentes métriques d'évaluation de chacun des algorithmes qu'on a déjà implémenter : Recall, Precision, F1 Score, Area Under ROC, Area Under PR.



```
▶ ▶≡ M↓  
results
```

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Logestic Regression	0.429814	0.891753	0.580050	0.987842	0.478709
Decision Tree Classifier	0.658385	0.944742	0.775988	0.788542	0.439176
Random Forest Classifier	0.440994	1.000000	0.612069	0.983306	0.739144
Gradient-Boosted Tree Classifier	0.665839	0.981685	0.793486	0.993399	0.793934
Naive Bayes	0.555280	0.011251	0.022054	0.506503	0.001276



Maintenant qu'on a mesurer les performances de nos modèles, on peut choisir le meilleur d'entre eux, selon les critères dont on a besoin.

Si par exemple on veut le modèle le plus précis on se base sur la précision :

▶ ▶ ML

```
df.sort_values(["Precision"], ascending=False)
```

	Unnamed: 0	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
2	Random Forest Classifier	0.440994	1.000000	0.612069	0.983306	0.739144
3	Gradient-Boosted Tree Classifier	0.665839	0.981685	0.793486	0.993399	0.793934
1	Decision Tree Classifier	0.658385	0.944742	0.775988	0.788542	0.439176
0	Logestic Regression	0.429814	0.891753	0.580050	0.987842	0.478709
4	Naive Bayes	0.555280	0.011251	0.022054	0.506503	0.001276

Dans ce cas, les meilleurs algorithmes sont : Random Forest avec une précision de 100%, le Gradient Boosted Tree avec 98% puis les arbres de décision avec une précision de 94%.

Selon le F1 Score :

▶ ▶ ML

```
df.sort_values(["F1 Score"], ascending=False)
```

	Unnamed: 0	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
3	Gradient-Boosted Tree Classifier	0.665839	0.981685	0.793486	0.993399	0.793934
1	Decision Tree Classifier	0.658385	0.944742	0.775988	0.788542	0.439176
2	Random Forest Classifier	0.440994	1.000000	0.612069	0.983306	0.739144
0	Logestic Regression	0.429814	0.891753	0.580050	0.987842	0.478709
4	Naive Bayes	0.555280	0.011251	0.022054	0.506503	0.001276

Encore cette fois le Gradient Boosted Tree est le meilleur avec un F1 Score de 80%.

Selon le Area Under ROC :

▶ ▶≡ M↓

```
df.sort_values(["Area Under ROC"], ascending=False)
```

	Unnamed: 0	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
3	Gradient-Boosted Tree Classifier	0.665839	0.981685	0.793486	0.993399	0.793934
0	Logestic Regression	0.429814	0.891753	0.580050	0.987842	0.478709
2	Random Forest Classifier	0.440994	1.000000	0.612069	0.983306	0.739144
1	Decision Tree Classifier	0.658385	0.944742	0.775988	0.788542	0.439176
4	Naive Bayes	0.555280	0.011251	0.022054	0.506503	0.001276

Et finalement Selon Area Under Precision-Recall :

▶ ▶≡ M↓

```
df.sort_values(["Area Under PR"], ascending=False)
```

	Unnamed: 0	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
3	Gradient-Boosted Tree Classifier	0.665839	0.981685	0.793486	0.993399	0.793934
2	Random Forest Classifier	0.440994	1.000000	0.612069	0.983306	0.739144
0	Logestic Regression	0.429814	0.891753	0.580050	0.987842	0.478709
1	Decision Tree Classifier	0.658385	0.944742	0.775988	0.788542	0.439176
4	Naive Bayes	0.555280	0.011251	0.022054	0.506503	0.001276

D'après ce qui précède on peut conclure que pour ce jeu de données, le meilleur modèle à utiliser est le **Gradient Boosted Tree**.

L'algorithme Naïve Bayes, ne donne pas de bons résultats (1% de précision), on peut expliquer cela par le déséquilibre qu'on a entre les deux classes (0.13% de la classe 1 et 99.87% non-fraude). Les algorithmes d'apprentissage automatique fonctionnent généralement mieux lorsque les différentes classes contenues dans l'ensemble de données sont plus ou moins égaux. S'il y a peu de cas de fraude, alors il y a peu de données pour apprendre à les identifier. Et c'est l'un des principaux défis de la détection de la fraude bancaires.

## Conclusion :

Enfin, nous nous sommes focalisés sur le Supervised Learning où on a les données sont étiquetées, mais il peut être intéressant de travailler avec d'autres techniques de modélisation comme le Unsupervised Learning ou encore le Deep Learning pour traiter ce problème de détection de fraude.

# Références bibliographiques :

- [1] Nick Pentreath, « Machine Learning with Spark », Packt Publishing, Year : 2014
- [2] Machine Learning Library (MLlib) Guide,  
<https://spark.apache.org/docs/latest/ml-guide.html>, Last retrieved on June 2021
- [3] Conference Paper : IEEE Big Data 2017, « Big Data Machine Learning using Apache Spark MLlib », [https://www.researchgate.net/publication/321149692\\_Big\\_Data\\_Machine\\_Learning\\_using\\_Apache\\_Spark\\_MLlib](https://www.researchgate.net/publication/321149692_Big_Data_Machine_Learning_using_Apache_Spark_MLlib)
- [4] Edgar Lopez-Rojas, « Synthetic Financial Datasets For Fraud Detection », Synthetic datasets generated by the PaySim mobile money simulator,  
<https://www.kaggle.com/ealaxi/paysim1>, Last retrieved on April 2021
- [5] Cem Dilmegani, « Synthetic Data Generation : Techniques, Best Practices & Tools », <https://research.aimultiple.com/synthetic-data-generation/> January 13, 2021, Last retrieved on April 2021
- [6] Dinara Rzayeva, Saber Malekzadeh, « Fraud Detection on Credit Card Transactions Using Machine learning », [https://www.researchgate.net/publication/347487399\\_Fraud\\_Detection\\_on\\_Credit\\_Card\\_Transactions\\_Using\\_Machine\\_learning](https://www.researchgate.net/publication/347487399_Fraud_Detection_on_Credit_Card_Transactions_Using_Machine_learning), December 2020, Last retrieved on May 2021
- [7] Apache Spark, <https://spark.apache.org/downloads.html>, Last retrieved on April 2021
- [8] Karlijn Willems, Apache Spark Tutorial : ML with PySpark, <https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning>, Last retrieved on May 2021